Symbolic verification of security aware workflows

Clara Bertolissi¹,² and Silvio Ranise²

¹LIF-CNRS, UMR 7279 & AMU, Marseille, France
²FBK (Fondazione Bruno Kessler), Trento, Italy

Context

Workflow management used in several applications
- E-business
- E-health
- E-government
- ...

Workflow management specification
- What are the tasks?
- What is the order of execution of the tasks?
- Which data are manipulated by each task?
- Who performs the tasks?
What are the tasks?
- register insurance claim
- check A of insurance policy
- check B of damage reported
- assess the results of checks A and B
- approve the payment of damage
- reject the payment of damage

What is the order of execution of the tasks?

Who performs the tasks?
- Three roles: Customer Service, Specialist A, Specialist B
- Six users: Anna, Adam, Benn, Beate, Carol, Chris

Role-Based Access Control (RBAC)
Additional authorization constraints:
Separation/Bound of Duty (SoD/BoD)

- **SoD**: If amount is larger than 5 KEuros, then the same user cannot execute both tasks check A and check B
- **BoD**: Task reject have to be performed by the same user who performed the task register

Which data are manipulated by each task?

- custID: unique identifier for customer
- type: enumerated data-type for identifying type of damages
- amount: money requested for damage
- answA, answB: either “ok” or “nok”
- decision: either “grant” or “refuse”
Workflow Satisfiability Problem (WSP)

- WSP consists of
  - checking if there exists an assignment of users to tasks such that
  - a security-aware workflow successfully terminates
  - while satisfying all authorization constraints.

- The runtime version of the WSP consists of
  - answering sequences of user requests at execution time
  - and ensuring successful termination
  - together with satisfaction of authorization constraints.

We propose a methodology to synthesize monitors for security-aware workflows based on the use of Satisfiability Modulo Theories (SMT) techniques.

Satisfiability Modulo Theories (SMT)

- Satisfiability: the problem of determining whether a formula expressing a constraint has a solution.

- The most well-known constraint satisfaction problem is SAT: the goal is to decide whether a formula over Boolean variables, formed using logical connectives, can be made true by choosing true/false values for its variables.
Some problems need more expressive logics such as first-order logic.

A first-order formula is formed using logical connectives, variables, quantifiers, function and predicate symbols.

A solution is an interpretation for the variable, function and predicate symbols that makes the formula true.

In satisfiability modulo theories (SMT), the interpretation of some symbols is constrained by a background theory.

A theory is a collection of facts over a signature $F$.

For example, let be the signature containing

- the symbols 0, 1, +, − and $<$, and

$Z$ be the structure that interprets these symbols in the usual way over the integers,

then the theory of additive arithmetic is the set of first-order sentences that are true in $Z$.

Other theories: theory of arrays, uninterpreted functions, bit-vectors...
SAT is NP-complete but first-order logic is undecidable.

Due to this high computational complexity, it is infeasible to build a procedure that can solve arbitrary SMT problems.

Most procedures focus on problems that occur in practice. They rely on the assumption that, although potentially big, most formulas produced by verification and analysis tools are shallow.

Today SMT solvers are very fast and can solve huge SMT problems (Have a look at http://www.smt-lib.org for an overview of the field)

---

Back to workflows: modeling issues

- control-flow semantics of workflows: Safe Petri nets
- model finite but unknown number of
  - workflow instances
  - users
- express
  - authorization constraints
  - data structures to model data-flow
Example

- **p** place: workflow state
- **o** token in place: current state
- **t** transition: workflow task
- **n** SoD constraint

Example of symbolic specification

State array variables in each component:

\[
\begin{align*}
p_1, \ldots, p_7, d, t_1, \ldots, t_5 & \quad a_1, \ldots, a, t_5, t_1, by, \ldots, t_5, by
\end{align*}
\]

Initial formulas:

\[
\begin{array}{|c|c|}
\hline
P(a_1) \quad \forall x & \begin{align*}
& p_0[x] \land \neg p_1[x] \land \neg p_2[x] \\
& \land \neg p_3[x] \land \neg p_4[x] \land \neg p_5[x] \\
& \land \neg p_6[x] \land \neg p_7[x] \land \neg d_{t_1}[x] \\
& \land \neg d_{t_2}[x] \land \neg d_{t_3}[x] \\
& \land \neg d_{t_4}[x] \land \neg d_{t_5}[x]
\end{align*} \\
\hline
U(a_2) \quad \exists x & \neg (a_{t_1, by}[0] \land \cdots \land \neg a_{t_5, by}[0]) \\
\end{array}
\]

Goal formula \( U \)

\[
\begin{array}{|c|c|c|}
\hline
U^1(a_1) \quad \exists x & \begin{align*}
& \neg p_0[x] \land \neg p_1[x] \land \neg p_2[x] \\
& \land \neg p_3[x] \land \neg p_4[x] \land \neg p_5[x] \\
& \land p_7[x] \land \neg d_{t_1}[x] \\
& \land d_{t_2}[x] \land d_{t_3}[x] \land d_{t_4}[x] \land d_{t_5}[x]
\end{align*} & U^2(a_2) \quad true
\end{array}
\]

\[
\begin{align*}
& \neg p_0[x] \land \neg p_1[x] \land \neg p_2[x] \\
& \land \neg p_3[x] \land \neg p_4[x] \land \neg p_5[x] \\
& \land p_7[x] \land \neg d_{t_1}[x] \\
& \land d_{t_2}[x] \land d_{t_3}[x] \land d_{t_4}[x] \land d_{t_5}[x]
\end{align*}
\]
The safety problem is to establish whether there exists a natural number $n$ such that the formula

$$I(a_1^0, ..., a_n^0) \land \bigwedge_{k=0}^{n-1} \tau(a_1^{k+1}, ..., a_n^{k+1}) \land U(a_1^n, ..., a_n^n)$$

is satisfiable. If so, there exists a run (i.e. a sequence of transitions) of length $n$ leading the system from a state in $I$ to a state in $U$. 
A general approach to solve instances of the safety problem is based on the symbolic computation of the set of backward reachable states.

The procedure to establish if the goal formula $U$ is reachable is based on iteratively computing the set $BR(a_1; \ldots; a_n)$ of states from which it is possible to reach $U$, by applying finitely many times the transition $\tau$.

$BR_b(\tau; U)$ represents the set of states which are backward reachable from the states in $U$ in at most $b$ steps.

In order to stop computing formulae in the sequence $BR_b(t; U)$, there are two criteria.

1) check whether $BR_b(t; U)^I$ is $T$-satisfiable: in this case, there exists a finite sequence of transitions in $T$ that leads the system from an initial state in $I$ to a state in $U$.

2) check whether $BR_{b+1}(t; U) \Rightarrow BR_b(t; U)$ is $T$-valid: in this case, $BR_b$ is the fix-point of the sequence of $BR_i$'s.

In our case, the procedure is guaranteed to terminate under suitable (technical) hypotheses that are satisfied in practice for workflows applications.
An SMT solver can act as a monitor solving the run-time version of the WSP.

For a particular instance of the workflow, the SMT solver can answer queries of the form “can user u execute task t and guarantee the successful termination of the workflow?”
Example

Set of active Users:
- Alice (a), Bob (b) and Charlie (c)

Autorisation policy:
- Role based access control (RBAC) associating users to roles and permissions of task execution to roles

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>r1</td>
</tr>
<tr>
<td>a</td>
<td>r2</td>
</tr>
<tr>
<td>a</td>
<td>r3</td>
</tr>
<tr>
<td>b</td>
<td>r2</td>
</tr>
<tr>
<td>c</td>
<td>r3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>t1</td>
</tr>
<tr>
<td>r2</td>
<td>t2</td>
</tr>
<tr>
<td>r3</td>
<td>t3</td>
</tr>
<tr>
<td>r4</td>
<td>t4</td>
</tr>
<tr>
<td>r5</td>
<td>t5</td>
</tr>
</tbody>
</table>

Runtime monitoring

- First, the formula $BR$ describing the set of states backward reachable from $U$ is instantiated with the set of users that are active.

- Then, an assertion $A(u; t)$ encoding the fact that user $u$ executes task $t$ is conjoined to $BR^j$ if $A(u; t) \land BR^j$ is found satisfiable by the SMT solver.

- If $A(u; t) \land BR^j$ is found satisfiable by the SMT solver, then user $u$ is permitted to execute task $t$ and $BR^{j+1}$ is updated to $A(u; t) \land BR^j$.

- Otherwise, $u$ is forbidden to execute $t$ and $BR^{j+1}$ is set to $BR^j$. 
The correctness of the approach relies on the observation that

- BR₀ represents all states from which the workflow instance can terminate successfully and no task has yet been executed.

- A(u; t) ∩ BR[j represents all states from which the workflow instance can terminate successfully after the execution of t by u.

- The unsatisfiability of A(u; t) ∩ BR[j implies that no state exists from which the workflow instance can successfully terminate after the execution of t by u.

Bob asks the system for executing task t₁.

Can b execute t₁?

Alice

Bob

Charlie
The solver grants the execution since, according to the RBAC policy, Bob has role r3 which is allowed to execute task t1.

Can b execute t1?

Grant execution

Charlie

Charlie is denied the execution of t2 since his roles has no such privilege (only role r1 can execute t2).

Can c execute t2?

Deny execution
Instead, Alice can execute t2 since she has not executed t1 and, according to the RBAC policy, Alice is assigned role 1.

Example of successful execution:
(Bob, t1), (Alice, t2), (Charlie, t3), (Alice, t4), (Bob, t5)

Remark: even if she has the rights to do so, Alice would be denied to execute t1, since this would prevent the successful termination of the workflow:
(Alice, t1), (?, t2)
Model Checking and SMT techniques can be used to guarantee the enforcement of authorization constraints and the successful termination of security-aware workflows.

- Formally sound
- Parametric in the number of users.
- Flexible: multiple workflow instances and data flow can be taken into account.

For more details, see

[Bertolissi and Ranise, Frocos’13; Bertolissi and Ranise, ICITST’13]
S = workflow specification

U = goal formula

\[ S = \text{workflow specification} \]
\[ U = \text{goal formula} \]

\[ B = \text{formula describing the set of states backward reachable from U, which is also a fix-point} \]

**Backward reachability algorithm**

```plaintext
function BR(\( S, U \))
1. \( P \leftarrow U; B \leftarrow \bot \)
2. while not entail(\( P, B \)) do
3. if not emptypoint(\( I, P \)) then return unsafe;
4. \( B \leftarrow P \lor B \)
5. \( P \leftarrow \text{simplify}(\text{Pre}(\tau, P)) \)
6. end
7. return (safe, \( B \))
```

```plaintext
function BR(\( G \))
1. \( i \leftarrow 0; BR^0(\tau, G) \leftarrow G; K^0 \leftarrow G \)
2. if \( \text{SMT}(BR^i(\tau, K) \land I) = \text{sat} \) then return unsafe
3. repeat
4. \( K^{i+1} \leftarrow \text{Pre}(\tau, K^i) \)
5. \( BR^{i+1}(\tau, K) \leftarrow BR^i(\tau, K) \lor K^{i+1} \)
6. if \( \text{SMT}(BR^{i+1}(\tau, K) \land I) = \text{sat} \) then return unsafe
7. else \( i \leftarrow i + 1 \)
8. until \( \text{SMT}(\neg BR^{i+1}(\tau, K) \rightarrow BR^i(\tau, K)) = \text{unsat} \)
9. return safe
end
```